Feature Based Algorithm Configuration: A Case Study with Differential Evolution

Nacim Belkhir, Johann Dréo, Pierre Savéant, Marc Schoenauer

Feature Based Algorithm Configuration: a Case Study with Differential Evolution

Nacim Belkhir[1,2], Johann Dréo[1],
Pierre Savéant[1], and Marc Schoenauer[2]

[1]Thales Research & Technology, Palaiseau, France
[2]TAO, Inria Saclay Île-de-France, Orsay, France
{nacim.belkhir,johann.dreo,pierre.saveant}@thalesgroup.com
marc.schoenauer@inria.fr

Abstract. Algorithm Configuration is still an intricate problem especially in the continuous black box optimization domain. This paper empirically investigates the relationship between continuous problem *features* (measuring different problem characteristics) and the *best parameter configuration* of a given stochastic algorithm over a bench of test functions — namely here, the original version of Differential Evolution over the BBOB test bench. This is achieved by learning an empirical performance model from the problem features and the algorithm parameters. This performance model can then be used to compute an empirical optimal parameter configuration from features values. The results show that reasonable performance models can indeed be learned, resulting in a better parameter configuration than a static parameter setting optimized for robustness over the test bench.

Keywords: Empirical Study, Black-box Continuous Optimization, Problem Features, Algorithm Configuration, Empirical Performance Model, Differential Evolution

1 Introduction

Today, it is widely acknowledged that the quest of a universal black box optimization algorithm is vain, even if the *No Free Lunch Theorem* [17] has been questioned in the continuous framework [1]. However, many algorithms exist, more or less specific to different classes of optimization problems, and the new grail of optimizers has now turned toward Algorithm Selection, as formulated by Rice [15], or Algorithm Configuration, that can be considered as yet another (meta-)optimization problem [6]. In both cases, the choice (of an algorithm, or of the parameters of a given algorithm) is made w.r.t. the user's preference, aka a performance criterion (e.g., quality of the solution obtained in a given CPU cost, the smallest CPU cost to reach a given solution quality, the probability to reach a given quality, with given thresholds, etc.).

A first approach to Algorithm Configuration is to optimize this performance criterion once and for all using a specific algorithm, e.g., SMAC [8]. But this results in a single configuration, and even if several problems are used to compute the performance criterion, the generalization of the results to other problems might be problematic.

More recent approaches are based on a description of the objective function in some *feature space*, and try to learn a mapping from this feature space onto the space of parameter configurations of the algorithm at hand, based on examples of the behavior of several configurations on a training set of objective functions. And the most successful approach for learning such a mapping is to first learn an empirical model of the algorithm performance (that predicts the performance criterion for a given set of features and an algorithm configuration). When a new problem arises (i.e. a new set of features), finding the algorithm configuration that is predicted to have the best performance is then straightforward. This approach, initially proposed in [10], has demonstrated successful results in different combinatorial optimization domains [7, 9, 18].

In continuous domains, however, though several feature sets have been proposed [12, 13], and successfully demonstrated to accurately classify problem instances [3, 13], only Algorithm Selection problems have actually been tackled [3, 12–14].

The present work addresses the Algorithm Configuration problem for continuous domains, building an Empirical Performance Model (EPM) based on the problem features in continuous search spaces cited above. The approach is experimentally validated with the original version of Differential Evolution [16], that has few hyper-parameters, but is known to be highly sensitive to their setting. The set of objective functions used for this validation is the well-known BBOB test bench [5].

The paper is organized as follows. Section 2 presents the general idea of feature-based Empirical Performance Model and subsequent algorithm configuration. Section 3 surveys the problem features in the case of continuous optimization. Section 4 then details the DE case study and the BBOB testbench used in this work. Section 5 describes the different experiments, and the corresponding results are detailed and discussed in Section 6, before the concluding Section 7.

2 Algorithm Configuration with an Empirical Performance Model

Context and notations The general context is the Black Box Optimization of *objective functions* $f : \Omega \mapsto \mathbb{R}$. An algorithm $\boldsymbol{\mathcal{A}}$ is given together with its control parameters $\theta \in \boldsymbol{\Theta}$. We assume that the objective functions can be described by some *features* $\boldsymbol{\psi} \in \boldsymbol{\Psi}$. The goal of Instance-based Algorithm Configuration is to find automatically, for a given objective function f described by its features ψ^f, the best possible *configuration* of $\boldsymbol{\mathcal{A}}$, i.e., values $\theta^*_f \in \boldsymbol{\Theta}$ such that running $\boldsymbol{\mathcal{A}}$ with parameters θ^*_f on f leads to optimal performances w.r.t. a given *performance measure* φ.

Empirical Performance Model The first step is to build an *Empirical Performance Model* (EPM) $\widehat{\varphi}$ that approximate φ on $\boldsymbol{\Psi} \times \boldsymbol{\Theta}$. $\boldsymbol{\mathcal{A}}$ is run to optimize different functions f_i (described by their features ψ_{f_i}) using different parameter configurations θ_j. This allows to compute the exact values $\varphi(\psi_{f_i}, \theta_j)$ for different pairs (i, j)[1]. The set of all $((\psi_{f_i}, \theta_j), \varphi(\psi_{f_i}, \theta_j))$ is a training set that can be used as input to any standard regression method to learn a model $\widehat{\varphi}$ for φ. Note that building such a model is done once, and hence its computer cost is not a critical issue.

[1] The same θ_j need not have been tried for all f_i.

Empirical Optimal Configuration When a new objective function g is to be optimized with $\mathcal{A}$, its features ψ_g are computed, and the optimization of $\widehat{\varphi}(\psi_g, \theta_g)$ on parameter space $\boldsymbol{\Theta}$ leads to the empirical optimal parameters of $\mathcal{A}$ for g. Here, the cost of computing the features ψ_g, in terms of number of calls to g, is here of utter importance. In particular, it should be compared to the cost of running a full 'ad hoc' meta-optimization of $\mathcal{A}$ parameters for g (using e.g., SMAC [8]).

The remaining of the paper is concerned with objective functions defined on some continuous domain $\mathcal{D} \subset \mathbb{R}^d$ for a given dimension $d \in \mathbb{N}$. Different features, taken from the literature, will first be discussed, before the case study is detailed.

3 Features for Continuous Optimization

In this section, a single objective function $f : \mathbb{R}^d \mapsto \mathbb{R}$ is considered, and the feature space $\boldsymbol{\Psi}$ is a vector space of p real-valued features. The black-box context implies that features should be computed from samples of f[2], i.e. n pairs $(x_i, f(x_i)))$, specifically gathered for that purpose. The set of values $\{f(x_i)|i = 1, \ldots, n\}$ is denoted $\mathcal{Y}$.

A first set of 55 features is taken from [12]. These features are grouped into six classes: the 3 y*−Distribution* features are related to the distribution of the values in $\mathcal{Y}$, the 18 *Levelset* features to the relative position of $\mathcal{Y}$ w.r.t a given threshold, the 9 *Meta-Model* features rely on meta-modeling of the sample set w.r.t linear and quadratic regression models, the 14 *Curvature* features on some numerical estimation of the Hessian and gradient of the problem, the 4 *Convexity* features on the empirical probability of convexity, and the 7 *Local Search* features on the ratio of local optima and global optima, estimated using some iterated local search procedure.
The y*−Distribution*, the *Levelset* and the *Meta-Model* features can all be evaluated on the same sample dataset, hence their cost altogether is n, the number of samples. However, some additional evaluations are required for the other feature classes, that depend on the previous samples. The orders of magnitudes are about $10^3 \times d$ for the *Convexity* features, around $10^4 \times d$ for the *Curvature* and the *Local Search* features.

A set of 16 *Dispersion* features was originally proposed in [11]. They are based on comparisons of the distances between best samples from different percentiles of the overall sample (in terms of solution quality) to the mean or median distance between all samples. Finally, 5 *Information Content* features were proposed in [13], giving information about the global structure of the landscape.

Recent works [3,12,13] successfully demonstrated that these features could be used in order to classify the optimization problems w.r.t their classes in BBOB (that will be introduced in Section 5) for the Algorithm Selection Problem. More recently, in [4] a subset of these features were used in order to improve the process of a parameter tuning algorithm, relying on the SMBO method [8].

[2] d, the dimension of the search space, can be considered as the only external feature — or the Algorithm Configuration can be conducted anew for each dimension (more in Section 5).

4 A Case Study in Continuous Domain: DE on BBOB

Differential Evolution and its Parameters Differential Evolution [16] is a popular continuous optimization algorithm that encountered many successes. It is also known for its simplicity, at least in its original version, that comes at the price of a large sensitivity to its parameter setting: this is the reason why it has been chosen here, making it easier to see big differences of results for different parameter settings. Several advanced versions of DE exist, that clearly outperform the original version, but comparing our results with theirs is left for further work.

DE generates new individuals from the current population by adding to each individual in turn a difference vector between two other individuals, and recombining the result with another individual from the population. The original version of DE has only four static parameters:

- the population size $\mathbf{NP} \in \mathbb{N}$;
- the strategy $\boldsymbol{\mathcal{S}} \in \{\mathrm{best1bin}, \mathrm{randtobest1bin}, \mathrm{best2bin}, \mathrm{rand2bin}, \mathrm{rand1bin}\}$ controls how to choose the endpoints of the difference vector;
- $\mathbf{F} \in [0, 2]$ controls the intensity of the difference vector;
- the crossover rate $\mathbf{CR} \in [0, 1]$.

In this work the population size $\mathbf{NP}$ is kept to the default value $15 \times d$ recommended by the authors[3]. Note that the recommendation for the other parameters is $\boldsymbol{\mathcal{S}} = best1bin$, $\mathbf{F} = 0.8$, and $\mathbf{CR} = 0.9$, and will be used as one of the baseline (Section 6).

Test Bench The following experiments consider the noiseless test functions from the Black Box Optimization Benchmark (BBOB[4]) [5]. The BBOB test bench is made of 24 analytically defined functions defined on $[-5, 5]^d$, with known global optima and known difficulties (e.g., non-separability, nulti-modality, etc). They have been manually classified in five classes of problems. In this work, only three of the BBOB classes are considered: 5 separable functions, 4 uni-modal functions with low or moderate conditioning, and 5 uni-modal functions with high conditioning (functions F1 to F14). Dimensions $2, 3, 5, 10$ are considered for all functions. As advocated in the original framework, any independent run on a function is actually done on a variant, in order to get over a possible algorithm bias. Variants are obtained from the original function by a translation of the position of the optimum and — for the non-separable functions — a rotation of the coordinate system.

Performance Measure Following the COCO/BBOB framework, the performance measure used here is the Expected Run Time[5] (ERT) needed to reach the optimal objective value with a given precision. Let $\overline{RT_s}$ be the average running time of successful runs, and p_s the empirical probability of success (out of the 15 independent runs). The ERT is defined as $ERT = ERT(f, \theta) = \overline{RT_s}/p_s$ if the results were obtained with DE configuration θ optimizing test function f.

[3] `http://www1.icsi.berkeley.edu/~storn/code.html`

[4] `http://coco.gforge.inria.fr`

[5] Measured as the number of function evaluations.

Features From the features briefly introduced in Section 3, different set of problem features are considered. All features have been computed using the R package kindly publicly provided by Pascal Kerschke[6]. Features are distinguished by their costs:

- $\boldsymbol{\psi}^{\star}$ includes all features from Section 3, with an initial sample of size $k = 2000 \times d$. However, as discussed in Section 3, the actual cost is much larger because of features requiring additional evaluations.
- $\boldsymbol{\psi}^{\bullet}_{\mathbf{k}}$: $k \times d$ is the size of the initial sample, and only features not requiring any additional function evaluation beyond those of the initial sample are considered (i.e., *Meta-Model*, *Information Content*, *Levelset*, y*–Distribution* and *Dispersion*). Results with $k = 500$ or $k = 2000$ are presented in the following.

Regression Model Preliminary experiments, not discussed here, lead to consider a Random Forests regression model (in accordance with [9]). A grid search with ten-fold cross-validation has been run on the meta-parameters of the Random Forest. The implementation of the scikit-learn python library has been used throughout this work[7], using 10 trees of maximal depth 200.

5 Experiments

Dataset A 40-steps discretization is used for $\mathbf{F} \in [0, 2[$ and $\mathbf{CR} \in [0, 1]$, resulting in $5 \times 40 \times 40$ different configurations. For each of the 14 functions of the test bench and for each dimension $d \in \{2, 3, 5, 10\}$, each one of its 15 variants is optimized with these 8000 DE configurations, and the ERT is computed. The initial dataset is hence made of 14×8000 entries per dimension, or 440 000 entries in total, considering all dimensions.

Dimensions As discussed, the dimension d can be considered as a particular feature, available "for free" (without any function evaluation), or as part of the problem definition — and there are as many problems as dimensions. These two points of view will be compared here: the EPM will be learned either using only the entries of the dataset of the same dimension, or all entries, and the dimension will then be used as an additional feature in the feature vector.

Cross-validation All experiments are based on a leave-one-out procedure: one of the 14 functions in the test bench is completely removed from the dataset (all dimensions and, of course, all variants). An EPM is then learned, and the left-out function considered a "unknown". The only exception is the *robust* baseline described below.

Baselines Different DE configurations are computed for each function, and used as a baseline for comparison with the results of the proposed approach. The *default configuration* recommended by DE authors (Section 4) is the first obvious baseline. However, it is likely to perform poorly across the whole test bench.

At the other extreme, the specific configuration found by some meta-optimizers for a given problem is likely to give the best overall results: two such meta-optimization

[6] `http://github.com/flacco`
[7] `http://scikit-learn.org/`

were performed for each function: on the one hand, the best configuration encountered while computing the full dataset using the grid described above is saved; on the other hand, SMAC [8] is applied to each function, using the ERT performance measure as fitness. The best of both configurations is reported as *adhoc configuration*.

Finally, one single SMAC optimization is performed using the average ERT over all 14 functions as performance: the idea behind this is to try to find some robust configuration that would give good results on all functions simultaneously. The resulting configuration is termed *robust* and considered as the reference (Section 6).

Experiment Costs All DE runs were allowed a maximum budget of $10^4 \times d$ function evaluations — though of course some runs did stop earlier, having reached the target precision. And all results have been averaged over 15 independent variants for each function and dimension.

Furthermore, both adhoc baseline configurations have the same cost, because the budget given to SMAC was purposely chosen to match that of the grid search, i.e., 8000×15 runs of at most $\times 10^4 \times dim$ evaluations each.

On the other hand, the robust configuration did cost 14 times more, as each of its iterations required to evaluate all the 14 functions — but has to be done only once.

Compared to that, the cost of finding the best empirical configuration using an EPM is the cost of the features: 500 or 2000 in the case of $\psi^{\bullet}_{500}$ and $\psi^{\bullet}_{2000}$ features, or around $2.10^4 \times d$ for the full set of features $\psi^{\star}$.

6 Results

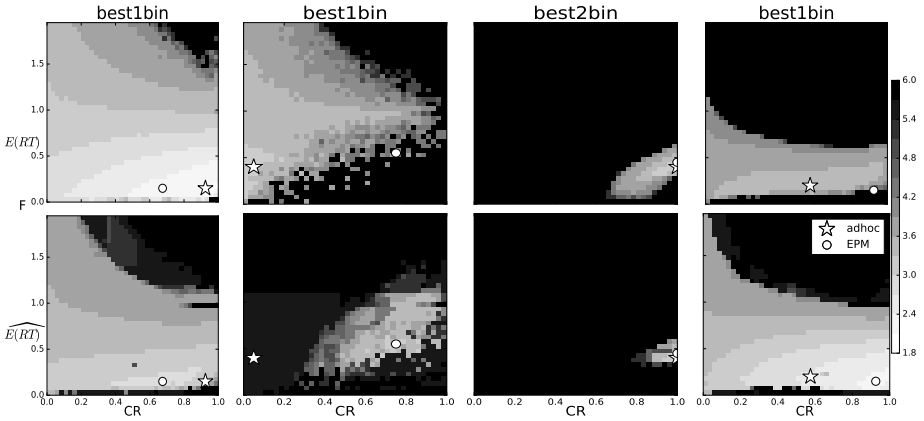
The series of experiments described above are presented in this section from two points of view: first, the different EPM are analyzed and compared to the ground truth — and the Empirical Optimal Configuration is compared to the true optimal configuration in parameter space. Then, the actual results of DE optimization using the Empirical Optimal Configuration are compared to those of the different baselines, keeping in mind the actual costs of the different approaches (see last paragraph above).

6.1 EPM Analyses

Due to space constraints, only few typical figures are displayed[8] (Figure 1) and will be discussed here. There are indeed some strong similarities between top and bottom colormaps for Figures 1a, 1c, and 1d, even though they correspond to different functions, dimensions, feature sets, and dimension handling modes. On the other hand, the two plots for F4 (Figure 1b) are very different, and here the EPM fails to capture even an approximate shape of the true ERT landscape.

But beyond such comparisons, the optimal configurations of both plots are shown (on both plots too, to ease the comparison), displaying very different situations: in Figure 1a and Figure 1c, both optima are rather close (1c), or at least are both in the same color area of the true ERT; on the opposite, in Figure 1b and Figure 1d, both optima

[8] Additional plots are available at `https://drive.google.com/open?id=0B9GuQcCjvwtFdkotR1h1N3dlOG8`



(a) g = F1, d = 5 (b) g = F4, d = 5 (c) g = F11, d = 10 (d) g = F5, d = 10

Fig. 1: Examples of comparisons between the true ERT (top) and the EPM (bottom) for 4 different functions and dimensions. Both EPMs of F1 (a) and F4 (b) have been learned only on samples from dimension 5, with features $\psi^{\bullet}_{k=2000}$ while those of F11 (c) and F5 (d) have been learned on samples of all dimensions, and with $\psi^{\bullet}_{k=500}$. Each subplot shows performances colormaps (without interpolation) of $\log_{10}(ERT/d)$, for one DE strategy, with the 2 other DE parameters F and CR on the axes. The true optimal configurations are plotted as white stars ☆ and the Empirical Optimal Configurations as white small circles ∘.

are far from one another, and the Empirical Optimal Configuration lies in a region of very poor true ERT: the performance of these configurations used within DE for the corresponding function will be poor too (see forthcoming Section 6.2).

6.2 Empirical Optimal Configurations at Work

For each computed EPM (described in Section 5), an Empirical Optimal Configuration is obtained by optimization on the parameter space, and the ERT of this configuration is obtained by running DE on each of the 15 variants of the target function. Tables 1a and 1b shows, for each function, the ratios of the ERTs of these different configurations against the *robust configuration* defined in Section 5 (the smaller the better). The first two columns are the other baselines, θ^d are the parameters values recommended by DE authors, and θ^L the adhoc configuration with best results (see Section 5). The two series of 3 columns correspond to the 3 different feature sets $\psi^{\bullet}_{500}$, $\psi^{\bullet}_{2000}$, and $\psi^{\star}$ (as defined in Section 4) and the two types of dimension handling (learning performed only on the same dimension as testing, or on all dimensions at once — see Section 5).

As it could be expected, the adhoc configuration is a clear winner, and the default values recommended by DE authors a clear loser. The results of dimension 5 (as well as those in dimensions 2 and 3, not shown due to space constraints) bring several good news: most proposed approaches perform better than the robust configuration, and at

g	θ^d	θ^L	train on dim 5 $\psi^\bullet_{500}$	$\psi^\bullet_{2k}$	$\psi^\star$	train on all dims $\psi^\bullet_{500}$	$\psi^\bullet_{2k}$	$\psi^\star$
F1	156	*17*	34	29	44	**26**	**26**	2.78
F2	153	*27*	57	**54**	**54**	57	**54**	240
F3	2415	*25*	158	639	*123*	158	129	∞
F4	561	*08*	97	97	**93**	97	97	∞
F5	139	*28*	214	232	**48**	54	54	∞
F6	145	*25*	71	**42**	49	56	71	486
F7	123	*37*	78	84	**62**	78	**62**	∞
F8	146	*29*	57	78	**36**	57	109	190
F9	114	*28*	48	66	**41**	98	73	133
F10	120	*27*	**35**	**35**	53	**35**	**35**	∞
F11	120	*29*	**30**	32	**30**	**30**	**30**	134
F12	113	*39*	171	137	**55**	118	137	1431
F13	118	*28*	**52**	91	∞	**52**	225	460
F14	119	*26*	∞	255	∞	∞	**85**	240

(a) Dimension 5

g	θ^d	θ^L	train on dim 10 $\psi^\bullet_{500}$	$\psi^\bullet_{2k}$	$\psi^\star$	train on all dims $\psi^\bullet_{500}$	$\psi^\bullet_{2k}$	$\psi^\star$
F1	94	*4.9*	**12**	14	42	**1.2**	**1.2**	3.4
F2	100	*7.7*	39	71	179	45	71	**32**
F3⋆	100	*0.2*	∞	∞	**1**	∞	∞	∞
F4⋆	10	*0.3*	∞	∞	**2**	∞	∞	∞
F5⋆	10	*0.2*	∞	∞	∞	∞	∞	∞
F6	157	*12.9*	33	**29**	46	56	82	56
F7	99	*0.22*	**24**	35	34	35	35	∞
F8	114	*17*	∞	∞	∞	∞	∞	**29**
F9	107	*15.4*	∞	28	29	27	29	**17**
F10	107	*12.9*	**17**	**17**	20	**17**	28	226
F11	114	*9.7*	**15**	16	**15**	**15**	16	16
F12	429	*7.8*	∞	21	**10**	∞	**10**	63
F13⋆	10	*0.1*	∞	2	∞	∞	∞	**1**
F14	100	*14.4*	∞	∞	**16**	∞	∞	28

(b) Dimension 10

Table 1: Percentages of the ERTs of different Empirical Optimal Configurations w.r.t. that of the *robust configuration* defined in Section 5, for dimensions 5 and 10. See text for details. Best results are printed in blue bold face when smaller than 100, in red italic when larger than 100; Worst results are printed in light gray; the $\star$ symbol indicates that the *robust configuration* never reached the target, and was artificially attributed an ERT of 15 times the maximum budget of one run, and ∞ indicates that the corresponding Empirical Optimal Configuration never reached the target.

least one does, for all functions but F3. For some functions (F11, and also F8 and F10), some feature-based approaches even get close to the best adhoc configurations, never being worse than twice that best performance, except for F4 — as could be foreseen on Figure 1b. When it comes to compare the different EPM settings, learning only from the single target dimension gives better results than learning for all dimensions together — and in the former case, using all available features does improve over only using the cheap features.

The situation is not so clear in dimension 10: in several cases, the Empirical Optimal Configuration cannot even reach the target in the allocated budget — a situation for which an example was given in Figure 1d. However, when an optimum can be found, similar conclusions to the dimension 5 case can be drawn, though not as contrasted.

6.3 Discussion

Our results suggest that the learned EPM can be similar to the actual performance map, at least in a large part of the parameter space. Nonetheless, when a particular feature is not included in the learning set, it can be very hard for the EPM to achieve good accuracy and performance prediction, as witnessed with function F4 (Figure 1b): only F3 and F4 are multi-modal functions, and they have very different structures (apart from being separable, and hence belong to the first BBOB class).

This was clear, too, with the F5 function (Figure 1d): F5 is the only linear function of the test bench, hence the EPM was learned without any linear function in the training set. However, the global accuracy of the EPM for F5 is rather good — not worse than F1 for instance (Figure 1a). But unfortunately, the small region of the parameter space where the EPM differs from the true ERT is the region that contains the optimal configuration.

This clearly demonstrates that a good accuracy over the parameter space of the EPM w.r.t. the true performances, such as the one being optimized by the learning algorithm (Random Forests here), is not required to reach the ultimate goal of the Algorithm Configuration process — find a quasi-optimal configuration for unknown instances. The only important property of the EPM is to be able to robustly identify good-performing regions of the parameter space. This opens several new possible research paths. At the level of the learning algorithm, the best regions of the parameter space could be weighted more than other parts of the space; at the extreme, rank-based learning could be used rather than regression of ERT values. At the level of the sampling, only good configurations could be used — e.g., the configurations encountered while running SMAC to find the true optimal configuration.

No clear differences can be seen between EPM learned using $\psi^{\bullet}_{500}$ or $\psi^{\bullet}_{2000}$, except for some functions, in dimension 10, where EPMs learned with $\psi^{\bullet}_{2000}$ solve the function while those learned from $\psi^{\bullet}_{500}$ don't. On the other hand, using the full set of features $\psi^{\star}$ does help, both in dimension 5 for the dimension-specific models, and in dimension 10 where it succeeds in reaching the target precision where the other models fail (e.g., F3 and F4, the only multi-modal functions of the test bench). While not surprising, this demonstrates that both the training set and the set of features should cover all the foreseeable difficulties of the unknown forthcoming instances. Any limited test bench (including BBOB) might hence be insufficient to learn a general-purpose configurator.

Finally, the fact that learning the EPM for a specific dimension leads to better results was to be expected. While this makes difficult to build an universal EPM, it does not prevent from any practical use of this approach, as the dimension is usually known (and constant) in most real world applications.

7 Conclusion

This paper has investigated the computation and use of an Empirical Performance Model (EPM) in the context of continuous black box optimization and has demonstrated that it is possible to learn a reasonable approximation of the real performance. More importantly, it was demonstrated that an efficient parameter configuration can be extracted from the learned EPM by optimizing the predicted performance, given a set of features on a new unknown function. In particular, it was possible to obtain empirical configurations that outperform a static parameter setting optimized for an average performance, over the whole test bench at the same overall cost. However, some open issues remain related to the robustness of the results, and deeper analyses are necessary to better understand (and avoid) some rare cases where the approach fails.

Several paths for further research are suggested by this work, both at the level of the learning algorithm and of the sampling of the parameter search space, as discussed in

Section 6.3. A promising direction is to embed the EPM as a parameter control mechanism within the optimization process itself, assuming that the features can be efficiently approximated using a rather small number of samples, (e.g., w.r.t. an approximation of the objective function, as proposed in [2]). This would open a new perspective on the on-line parameter tuning grail.

References

1. Auger, A., Teytaud, O.: Continuous Lunches are Free plus the Design of Optimal Optimization Algorithms. Algorithmica (2009), https://hal.inria.fr/inria-00369788
2. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Surrogate Assisted Feature Computation for Continuous Problems. In: to appear in Proc. LION 10 (2016), https://hal.archives-ouvertes.fr/hal-01303320, to appear
3. Bischl, B., Mersmann, O., Trautmann, H., Preuß, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation. pp. 313–320. ACM (2012)
4. Bossek, J., Bischl, B., Wagner, T., Rudolph, G.: Learning feature-parameter mappings for parameter tuning via the profile expected improvement. In: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference. pp. 1319–1326. ACM (2015)
5. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. Tech. Rep. RR-7215, INRIA (2010)
6. Hoos, H.H.: Programming by optimization. Comm. of the ACM 55(2), 70–80 (2012)
7. Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K.: Performance prediction and automated tuning of randomized and parametric algorithms. In: CP, pp. 213–228. Springer (2006)
8. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. LION 5, pp. 507–523. Springer (2011)
9. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence 206, 79–111 (2014)
10. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Principles and Practice of Constraint Programming-CP 2002. pp. 556–572. Springer (2002)
11. Lunacek, M., Whitley, D.: The dispersion metric and the cma evolution strategy. In: Proc. 8th GECCO. pp. 477–484. ACM (2006)
12. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proc. 13th GECCO. pp. 829–836. ACM (2011)
13. Munoz, M., Kirley, M., Halgamuge, S.K., et al.: Exploratory landscape analysis of continuous space optimization problems using information content. Evolutionary Computation, IEEE Transactions on 19(1), 74–87 (2015)
14. Muñoz, M.A., Kirley, M., Halgamuge, S.K.: A meta-learning prediction model of algorithm performance for continuous optimization problems. In: PPSN XII, pp. 226–235. Springer (2012)
15. Rice, J.R.: The algorithm selection problem. Advances in Computers 15, 65–118 (1976)
16. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11(4), 341–359 (1997)
17. Wolpert, D., Macready, W.: No free lunch theorems for optimization. Evolutionary Computation, IEEE Transactions on 1(1), 67–82 (1997)
18. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for sat. Journal of Artificial Intelligence Research pp. 565–606 (2008)